Learning more with the same effort: how randomization improves the robustness of a robotic deep reinforcement learning agent

Lucía Güitta-López[1*], Jaime Boal[1] and Álvaro J. López-López[1]

[1]Institute for Research in Technology (IIT), ICAI School of Engineering, Comillas Pontifical University, Santa Cruz de Marcenado, 26, Madrid, 28015, Madrid, Spain.

*Corresponding author(s). E-mail(s): lucia.guitta@iit.comillas.edu;
Contributing authors: jaime.boal@iit.comillas.edu; alvaro.lopez@iit.comillas.edu;

Abstract

The industrial application of Deep Reinforcement Learning (DRL) is frequently slowed down due to an inability to generate the experience required to train the models. Collecting data often involves considerable time and financial outlays that can make it unaffordable. Fortunately, devices like robots can be trained with synthetic experience through virtual environments. With this approach, the problems of sample efficiency with artificial agents are mitigated, but another issue arises: the need to efficiently transfer the synthetic experience into the real world (sim-to-real). This paper analyzes the robustness of a state-of-the-art sim-to-real technique known as Progressive Neural Networks (PNNs) and studies how adding diversity to the synthetic experience can complement it. To better understand the drivers that lead to a lack of robustness, the robotic agent is still tested in a virtual environment to ensure total control on the divergence between the simulated and real models. The results show that a PNN-like agent exhibits a substantial decrease in its robustness at the beginning of the real training phase. Randomizing

specific variables during simulation-based training significantly mitigates this issue. The average increase in the model's accuracy is around 25% when diversity is introduced in the training process. This improvement can translate into a decrease in the number of real experiences required for the same final robust performance. Notwithstanding, adding real experience to agents should still be beneficial, regardless of the quality of the virtual experience fed to the agent. The source code is available at: https://gitlab.com/comillas-cic/sim-to-real/pnn-dr.git

Keywords: Reinforcement Learning, Deep Learning, Sim-To-Real, Domain Randomization, Sample Efficiency, Robotics

1 Introduction

As stated by Sutton and Barto [1], Reinforcement Learning (RL) is an "approach that is much more focused on goal-directed learning from interaction than are other approaches to machine learning. [...] The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. [...] Exercising this connection (between the learner and the environment) produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals." RL and its deep-learning variant (DRL) are used in many disciplines because of their ability to tackle complex problems that involve sequential decision-making. Unlike supervised and unsupervised learning, which require previously collected (and labeled in the former case) datasets for training, RL learns by trial and error from the samples generated through the interaction between an intelligent agent and the environment. The general formulation of an RL problem is shown in Figure 1. In each step, the agent, which is the decision-maker, observes the current environment, state and the reward obtained and, following a policy, decides which action to perform next. The agent's final goal is to maximize the cumulative reward, named expected return [1].

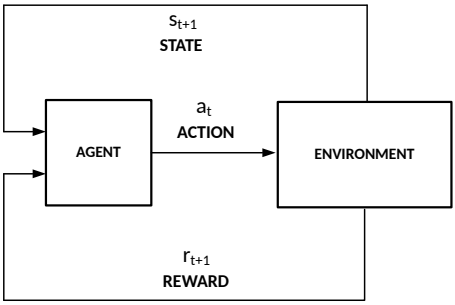

Figure 1: Interaction between the agent and the environment in an RL setting.

Robotic arms typically operate in a predefined and repetitive manner. Despite the large amount of experience still required, RL enables them to deal with stochasticity [2]. Generating that experience through trial and error in a physical setup could lead to collisions and result in structural damage to expensive assets. Modelling and training in a virtual environment with a realistic physics engine enables the agent to explore and exploit sub-optimal policies that could have undesirable consequences in a real scenario. This approach mitigates the low sample efficiency problem of DRL methods [3], but creates a new issue: how to efficiently transfer the synthetic experience to the real world. This is known as *the sim-to-real problem.*

To address this issue, we have drawn on the work about Progressive Neural Networks (PNNs) presented by Rusu et al. [4]. In this architecture, a single heavy neural model is first trained with synthetic experience (the virtual column) and then it is augmented via lateral connections with as many lightweight neural models as required to tackle real-world problems (the real columns). These interface models with the physical world are trained using real experience enhanced by the previous insights obtained from the virtual environment. We regard this approach as extremely promising for two main reasons: 1) it enables reliable representations of actual situations that can be generated with relatively few real experience samples; and 2) it makes it

possible to use the same set of synthetic data (i.e., virtual experience), to help a single agent master several real tasks.

To unlock the full potential of PNNs, thoroughly training the agent in the virtual environment is crucial. This paper attempts to determine the robustness of an RL agent right after the virtual training phase (i.e., before fine-tuning the weights of the real columns with real experience). In our rationale, an agent that exhibits high robustness to discrepancies between the virtual training environment and a modified version of it (i.e., sim-to-sim) should require significantly fewer real experiences at this stage than an agent highly sensitive to this effect. In particular, we analyze the effect on robustness of introducing randomness in a subset of the virtual environment parameters during training [5], and compare them with the fixed virtual model approach proposed in the seminal PNNs paper [4].

The main contributions of our work are: 1) an original benchmark that combines virtual and real experiences to measure the robustness of any artificial agent; and 2) the quantification of the robustness gain achieved in PNNs with Domain Randomization (DR), which can be interpreted as an efficiency increment in the sim-to-real process. These contributions pave the way for the hybridization of PNNs and DR that –to the top of our knowledge– is a topic yet to be explored.

The rest of the paper is structured as follows. Section 2 briefly summarizes several basic reinforcement learning concepts that readers familiar with the topic can safely skip. Section 3 provides an overview of the state-of-the-art of DRL in robotics, techniques to address the differences within real and simulated worlds, and RL robustness analysis techniques. Section 4 describes the method, whose results are presented and discussed in Section 5. Finally, conclusions and future work are put forward in Section 6.

2 Preliminaries

Reinforcement learning is an area of Machine Learning (ML), in which an agent learns from the feedback given by the environment during their interactions. The elements that define an RL problem are the environment's *state*, which the agent perceives as an observation $S_t \in S$, where S is the set of states; the *action* the agent performs $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions available in S_t; and the *reward* $R_{t+1} \in R \subset \mathbb{R}$, which is the result of the agent's decision and the environment's dynamics (Figure 1). The agent acts following a *policy* $\pi_t(a \mid s)$ that seeks to maximize the reward $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ received over time. $\gamma \in [0, 1]$ is the discount rate that modulates the effect of future rewards in the present moment, where k denotes the number of time steps, and t refers to the current time step.

RL problems can be formulated as a Markov Decision Process (MDP) [6] or a Partially Observable Markov Decision Process (POMDP) [7]. However, since the mathematical tools available to solve POMDPs do not usually scale well, these problems are converted to an MDP by means of a transformation function that derives the environment's state from observations [1]. A finite MDP is a 5-tuple (S, A, P, R, γ) that consists of a finite set of states S, a finite set of actions A, a state transition probability matrix P, a reward function R, and a discount factor γ.

RL model-free algorithms, where the agent only learns the policy that maximizes the reward by trial and error without prior information about the model dynamics or environment transitions, can be classified into two main groups: value-based and policy-based methods [3, 8]. The algorithm employed in this paper belongs to the latter category. Policy-based methods optimize the parameterized policy $\pi_\theta(a \mid s) = Pr[a \mid s;\ \theta]$ by either using gradient ascent or maximizing local approximations on the expected return. While value-based

strategies can be trained off-policy, optimization in policy-based algorithms is usually performed on-policy, using experience information obtained with the latest version of the policy.

In the Asynchronous Actor-Critic (A2C) [9] and Asynchronous Advantage Actor-Critic (A3C) [9] –which we consider a policy-based method, even though it is sometimes classified in literature as falling between both categories– the *actor* oversees policy optimization, while the *critic's* role is to estimate the value function, this is, the expected value for an agent to be in a specific state. The term asynchronous refers to the fact that the network is composed of multiple agents that operate individually, interact independently with their environment, and only share the results every n steps to update the global network. The reduction of the variance of the bootstrapped return's estimator required to obtain the policy gradient achieved with the A3C algorithm provides great results in a large span of case studies [4, 10, 11]. Hence, the A3C (Figure 2) is used in this paper due to its strengths for DRL problems that deal with discrete action spaces and its easy parallelization, which substantially decreases training times [12]. Please refer to Appendix A for a more detailed explanation of the algorithm.

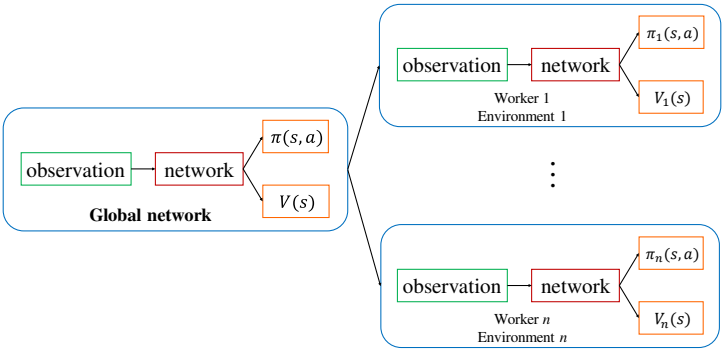

Figure 2: Schematic diagram of an A3C. Each worker trains and evaluates its network individually. After a given number of steps, the parameters are shared with the global network, which is updated with the set of parameters that lead to a better outcome.

3 Related work

Reinforcement learning agents have traditionally failed to scale to high dimensional spaces (e.g., when observations are images) [13]. Deep reinforcement learning (DRL) overcomes the memory and computational complexity limitations of RL methods thanks to the function approximation and representation learning properties inherent to deep neural networks [14]. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) [15, 16] enable such a broad feature abstraction level that, in practice, the need for feature engineering is eliminated.

DRL provides mechanisms for resolving complex robotics issues, including optimal control policies in high dimensional spaces, the interaction with dynamic environments, high-level task planning, and advanced manipulation, among others [17, 18]. Although DRL facilitates solving robotic tasks with its representation generally based on state-action discretization and Bellman's equation approximations [6], there are still several challenges. These hurdles include under-modeling, which involves failing to achieve an accurate virtual scenario, and model dynamics, that lead to poor performance in the real world, the reward design, the goal specification, and the collection of enough experience to assure an efficient learning [19].

The last issue is of utmost importance in DRL applied to robotics, since the collection of real experience is often unaffordable due to time and asset availability constraints. The general solution is to generate synthetic experience in a virtual setup and then transfer the optimal policies learned to the physical robot. This sim-to-real approach poses new challenges like how to account for the differences between the virtual and real environments, which literature has addressed with several different approaches [20].

The simplest method, fine-tuning, is slow for complex tasks and there is no guarantee of success. While fine-tuning is typically applied to the model's hyperparameters, Chen et al. [21] propose a method to automatically adjust the simulated environment to make it look more like the real setup. Relying on a theoretical demonstration, they introduce Latent Markov Decision Processes (LMDPs) as a set of MDPs with tunable parameters plus a distribution function over them. They argue that training an agent in a simulator made of these LMDPs should lead to an optimal policy that will achieve a good performance on the real environment, whose dynamics are already included in one of the MDPs. However, this proposal has two limitations: one is the lack of a practical implementation; and the other is the aforementioned assumption about the integration of the "true" MDP on the LMDPs, since it implies having some knowledge about the environment, something that is not always possible.

Other noteworthy techniques to facilitate the leap between the virtual and the real are Knowledge Distillation [22], Meta Reinforcement Learning (MetaRL) [23], Robust Reinforcement Learning [24], and Imitation Learning [25, 26]. [27] learns two tasks in simulation and then, using policy distillation, obtains a single policy that can be deployed into the real scenario. The drawback is that a dataset needs to be created for every virtually-trained agent, which leads to scalability issues as the number of tasks grows. [28] uses the MetaRL "learning to learn" idea to train a policy under different dynamic conditions in a virtual environment, and then tests it in the real setup. The disadvantage is that learning to learn optimal policies for all the MDPs that can be defined for a single problem can unnecessarily increase the complexity of simple tasks.

In transfer learning, Zero-Shot transfer [29] suggests training the model in a hyper-realistic simulator to enable direct deployment into the real scenario.

They propose a multi-stage RL strategy where the agent first learns the vision task using a disengaged environment representation (unsupervised learning), and then the acting part (reinforcement learning). This way, the trained agent is able to dissociate the elements from the scenario and deduce the policy for different environment configurations. Even though the results are quite favorable, the need of training two models, each of them with a different learning method, and its associated complexity in terms of data gathering and tuning are their main drawbacks. Domain Adaptation (DA), which succinctly translates data from a source domain to a target domain, is normally used for vision-based tasks like [30], whose approach is applicable to RL problems with continuous action spaces. However, there is also research surrounding the application of DA to generalize the policy learned. [31] utilizes synthetic data and DA to improve the results obtained when the agent is only trained with domain randomization. The input to its pixel-level DA model is a synthetic image from the simulator and, subsequently, they adapt the picture to make it more similar to the real environment. This technique uses Generative Adversarial Networks (GANs) [32] to adjust the images appearance, which means that, if the scenario changes, the GAN should be trained again. GANs are widely used as a DA tool to either adapt images from a virtual environment to make them resemble the real scenario, or to modify the characteristics of the environment features [33, 34].

An encouraging method for transferring the knowledge learned in a virtual environment is presented by Rusu et al. [4] as an extension of their work on Progressive Neural Networks [35], whose goal was to avoid catastrophic forgetting in sequences of tasks. Instead of relying on the rendering engine of the virtual environment or on heavy image processing, [4] suggests that an agent could be trained in simulation and, through lateral connections between

layers or blocks of layers, help other networks (named columns), trained in different virtual or real domains, learn to handle the differences or adapt to new tasks. In their experiments, the agent's objective was to reach a certain point in space with the grippers of a JACO robotic arm, first in simulation and then in the real domain. The input to the DRL algorithm, an A3C, is a raw 64 x 64 RGB image, and the outputs are the nine discrete joint policies and the value function. They performed different experiments varying the columns' network architectures and sizes, achieving 100% success in the real scenario with a sparse reward configuration and a recurrent neural network.

Due to these promising results, the model described in [4] is the inspirational framework adopted in our work. We kept the A3C as the learning algorithm, both due to its reliable outcomes obtained in DRL problems [4, 10, 11] and ease of parallelization, which is essential to speed up training times [12]. However, since our goal is to minimize the amount of experience needed for the real column, some modifications were made to our Baseline Model with respect to the original proposal (Appendix A).

Using this model as reference, we introduced Domain Randomization (DR) during training to improve the transfer to reality, that is, the robustness of the PNN approach. DR is a DA technique where training is conducted in a virtual domain whose attributes can change like they would in a real scenario. As demonstrated in [5], training models in simulated environments with enough variability can make the agent agnostic to the workspace. They trained a modified VGG-16 CNN [36] detector with rendered images generated with random camera and object poses, and variable lighting conditions and textures to mimic their physical setup. Without further training or parameter adjustment, they achieved 1.5 cm accuracy in the real world. [37] proposes a vision-based model where randomized images from the virtual scenario and

reality go through a GAN that translates them into a canonical view to feed the model. More recently, [38] improved model generalization for robotic manipulators by adding random unrealistic perturbations and implementing their intervention-based invariant transfer learning method (IBIT).

Finally, [39, 40] suggest methods to evaluate model performance and robustness during and after training. Considering it a classification problem, each episode is labeled as a success or failure, depending on whether or not the goal is reached, respectively. The metrics used to analyze and compare the trained models are the learning time, average accuracy, and maximum failure distance in evaluation.

4 Method

This section presents the general details of the research approach followed, the model design and training details, and the virtual test bench used to perform the robustness assessment cycles.

4.1 Problem definition

All the experiments presented in this paper were performed in a virtual setup that consists of an IRB 120 robotic arm by ABB[1], a fixed externally-mounted monocular camera, and a red 0.03 m cubic target the robot must reach within a maximum number of steps (an episode). The IRB 120 has 6 degrees of freedom (DoF), a reach of 0.58 m, a payload of 3 kg, and an armload of 0.3 kg. Its operating ranges and velocities are shown in Table 1.

The robot's behavior was simulated in MuJoCo [41] to accurately capture physics. Since rich visual textures were not considered at this stage of the research, the virtual environment was rendered using Matplotlib. Figure 3 presents examples of the images captured by the camera. Each of them is a

[1] https://new.abb.com/products/robotics/industrial-robots/irb-120/irb-120-data

Table 1: Working range and velocity of the IRB 120 axes.

Axis motion	Working range	Velocity
Axis 1 (rotation)	+165 to −165°	250 °/s
Axis 2 (arm)	+110 to −110°	250 °/s
Axis 3 (arm)	+70 to −110°	250 °/s
Axis 4 (wrist)	+160 to −160°	320 °/s
Axis 5 (bend)	+120 to −120°	320 °/s
Axis 6 (turn)	+400 to −400°	420 °/s

sample of the observations used to feed the model at every time step. The initial target and joint locations were randomly chosen.

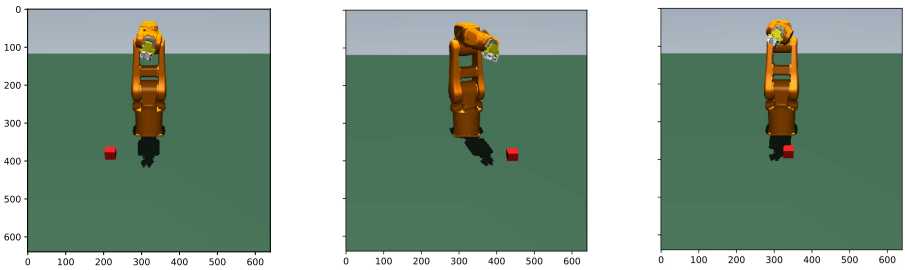

Figure 3: Sample input images provided to the model when the external camera is mounted at (180°, −30°). Initial target (the red cube) and joint positions are random. Axes indicate pixel coordinates.

4.2 Research approach

Figure 4 summarizes the methodology employed. On the top branch, an artificial agent that was used as the reference or Baseline Model (BM) controlled the robot according to the architecture of a regular PNN. Then, a virtual test bench measured the robustness of the BM to changes in the camera settings and, finally, the results were structured and analyzed. Subsequently, an analogous method was followed on the bottom branch to obtain the robustness results of an improved artificial agent in which the training phase was enriched by applying DR to the camera location (DR Model (DRM)). For the sake of

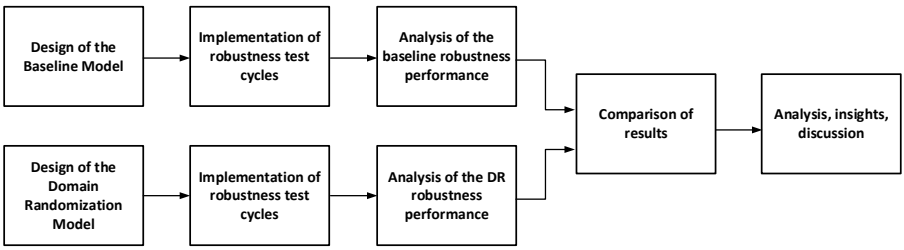

Figure 4: Schematic diagram of the methodology.

simplicity, only the camera pose was used to represent discrepancies between the virtual and the real worlds, acknowledging that the model is sensitive enough to changes in this variable. This is supported by Tobin et al. [5]. According to their results, the DR feature that most affects accuracy is the camera pose. Although our work cannot be directly compared with theirs since the problem addressed is different, as they already stated, it can be considered an extension because, besides detecting the target, we also infer the robot's pose from the image.

4.3 Model design and training

The agent was trained in episodes. At the beginning of each episode, the target and robot positions were randomly initialized following a uniform distribution constrained by the reachable workspace and image limits, which is 15% of the robot's operating range (Table 1). An episode ended after 50 steps or if the distance from the gripper to the target was less than or equal to 5 cm. The discount factor (γ) was set at 0.99 to ensure that the agent would take future rewards into account. In total, the agent gathered experience from 70 million steps. Every 50 thousand steps, training was halted to perform 40 evaluation episodes and compute the average, maximum, and minimum distances from the gripper to the target location, obtaining the curves to assess the effectiveness of the training and estimate sample efficiency, among other metrics.

As aforementioned, the BM was largely designed following the PNN proposal. However, significant effort was put into studying the effect of the MDP's action space and the design of the reward signal to acquire a better understanding of the drivers of control performance. Although we believe this part of the work is interesting, we also consider it outside the main scope of the present paper and, therefore, the details of the BM design, including the choice of the algorithm hyperparameters, have been relegated to Appendix A.

The camera pose was fixed for the BM during training on both axes: 180° for the z-axis and $-30°$ for the y-axis. This setup provided a close to optimal perspective of the robot and the target positions, as illustrated in Figure 3. While the DRM inherited the action space, reward signal, and hyperparameters of the BM, camera pose variability on both axes was also included during training. Specifically, the z-axis camera pose varied in the interval $[160°, 200°]$, and the y-axis in $[-40°, -20°]$.

Table 2 summarizes the training parameters of both models. WRLL stands for Working Range Lower Limit and WRUL for Working Range Upper Limit (Table 1), where $U(x, y)$ means this parameter was obtained by sampling a uniform distribution from x to y. To ensure a fair comparison between models, all parameters were identical except for the randomized variables in the proposed DR approach (i.e., those corresponding to the camera pose).

4.4 Evaluation: the virtual test bench

After the training phase, control robustness was assessed on a virtual test bench where almost any parameter can be modified. Using only the virtual column (i.e., right before including real experience in the lateral connections and the real columns of the PNN), the agent's policy was extracted from the neural network and evaluated against a different virtual model of the environment, which acted as a surrogate of the real model. This provided a

Table 2: Training parameters for the Baseline and the DR models.

Model	Millions of steps	Success distance[a]	Episode initial position[b]	Episode target position[c]	Episode camera z-axis	Episode camera y-axis
BM	70	5 cm	$U(+15\%\text{WRLL}, -15\%\text{WRUL})$	$x \sim U(0.2, 0.4)$ cm $y \sim U(-0, 3, 0.3)$ cm	180°	−30°
DRM	70	5 cm	$U(+15\%\text{WRLL}, -15\%\text{WRUL})$	$x \sim U(0.2, 0.4)$ cm $y \sim U(-0, 3, 0.3)$ cm	$U(160°, 200°)$	$U(-40°, -20°)$

[a]The success distance is measured from the robot's gripper to the target position.
[b]The robot's initial position is set according to a uniform distribution in which the minimum and maximum values are respectively 15% larger and 15% smaller than the robot's lower and upper working range limits.
[c]The episode target position is also sampled from uniform distributions, one for the x-coordinate and the other for the y-coordinate, whose boundaries represent the distance in cm from the base of the robot.

highly interpretable measurement of the effectiveness of the virtual training stage.

An episode was considered as successfully completed if the distance between the gripper and the target was no more than 10 cm, twice as long as before, in the belief that training the agent in more demanding conditions than those actually required should lead to better performance. To prevent overly optimistic results, the virtual test bench had a higher variability in the camera pose than the one used to train the DRM. The rest of the parameters remained unchanged.

The camera pose swept within the interval $[140°, 220°]$ for the z-axis and $[-50°, -10°]$ for the y-axis with a 5° granularity (Figure 5). The shaded region represents camera poses that were presented to the DR agent during training (anchored in the center for the Baseline), only one-third of the evaluation area. This parameterization let us measure how the agent performed when it interpolated within its experience and when it was forced to extrapolate. Figure 6 shows the limit poses of the camera to illustrate the viewpoint changes along each axis.

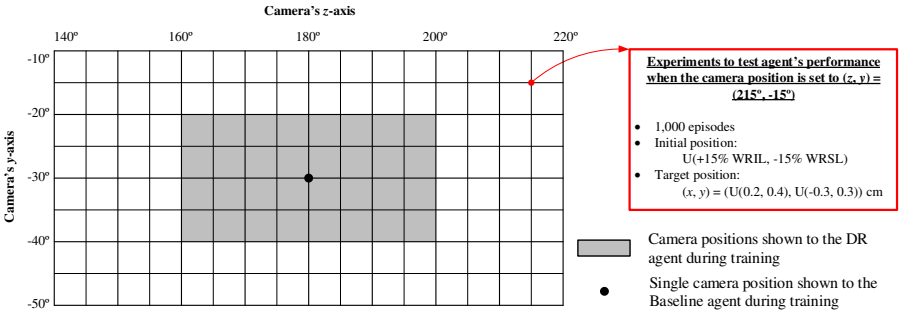

Figure 5: 5° granularity grid designed to evaluate the models. The black dot indicates the camera pose during training for the BM and the grey area the orientations presented to the DR agent.

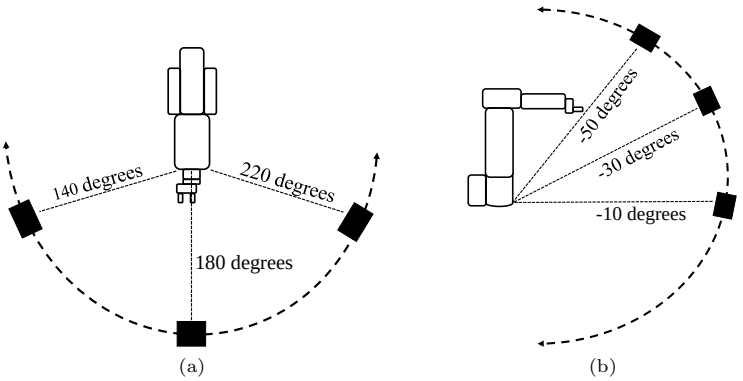

Figure 6: Range of possible camera orientations under evaluation around the z-axis (a) and y-axis (b). The radius of the sphere $r = 2\,\text{m}$ is constant.

Every camera location was evaluated for 1000 episodes, each with its own random initial joint and target positions, for a total of 153,000 combinations. To compare the models, the number of steps required to reach the steady-state during learning, the cumulative reward, the average accuracy, and the distance to the target in unsuccessful episodes were all considered (Table 3).

5 Results and discussion

This section presents the results for the BM, followed by those of the DRM. The same fixed scheme was repeated for both models to enhance understanding. The experiments were conducted on a PC running Ubuntu 20.04 equipped

Table 3: Metrics for model comparison.

Number of training steps	Cumulative reward[a]	Average accuracy[b]	Failure distance[c]
Steps to reach the steady-state regime	$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$	$\frac{S}{S+F}$	cm between the gripper and the target

[a] $\gamma \in [0, 1]$ = discount rate; k = number of time steps; t = current time step.
[b] We consider this a classification problem. Episodes are tagged successful (S) if the target is reached, or failed (F) otherwise.
[c] Only defined for unsuccessful episodes.

with an Intel Core i9-10900KF processor at 3.70 GHz, 64 GB of DDR4 RAM, an NVIDIA GeForce RTX 2080 Ti GPU, and a 2 TB M.2 SSD. The algorithms were programmed in Python 3.8 using PyTorch [42].[2]

5.1 Baseline Model (BM)

5.1.1 Training

Figure 7 depicts the average reward obtained during the evaluations, interleaved every 50,000 steps in the training phase (Section 4.3). The 70 million steps took around 14 hours to complete. Once the steady-state regime is reached in around 35 million steps, the model stops learning. This means that the robot is incapable of reaching the target with less error. A higher value is indicative of the robot's proficiency with the task, but it cannot be guaranteed until the individual cases are analyzed below. Note that around 40 million and 60 million steps, the agent seems to explore and exploit a sub-optimal policy, leading to a temporary decrease in the average reward. Even though the stability of the models, as well as its convergence, can be guaranteed only in tabular problems, the empirical results show that there are no relevant issues regarding the steadiness during training and evaluation.

[2] The source code is available at: https://gitlab.com/comillas-cic/sim-to-real/pnn-dr.git

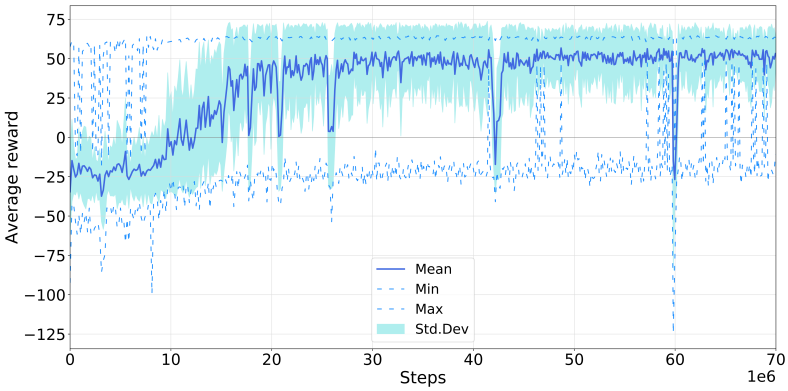

Figure 7: Average rewards obtained during the evaluation breaks interleaved in the 70 million episode training of the BM. The steady-state regime, with an average reward of roughly 50, begins at around 35 million steps. Note that within this period the agent exploited two sub-optimal policies at steps 42 million and 59 million.

5.1.2 Robustness

The outcomes are analyzed using a figure inspired by the orthographic projection system (Figure 8). The matrix, which corresponds to the floor view, is a heat map that presents the average accuracy obtained after evaluating the BM across all combinations of the z- and y-axes. On the graph, the average accuracy is displayed along the z-axis when the y-axis orientation is set to $-30°$. The left plot shows the average accuracy when the z-axis is kept at $180°$.

Although the Baseline agent was trained with a static camera at $(180°, -30°)$, the accuracy remains above 90% for a wider interval: $[165°, 195°]$ on the z-axis and $[-35°, -25°]$ on the y-axis. It even achieves 100% accuracy on the training viewpoint (i.e., the training curve stalled because the agent already mastered the task), and on two contiguous spots along the same $-30°$ height. The maximum failure distance is 48 cm, registered at $(140°, -25°)$ and at $(220°, -50°)$. Both poses correspond to unfavorable scenarios, where the camera is visibly below and above the initial y-axis orientation.

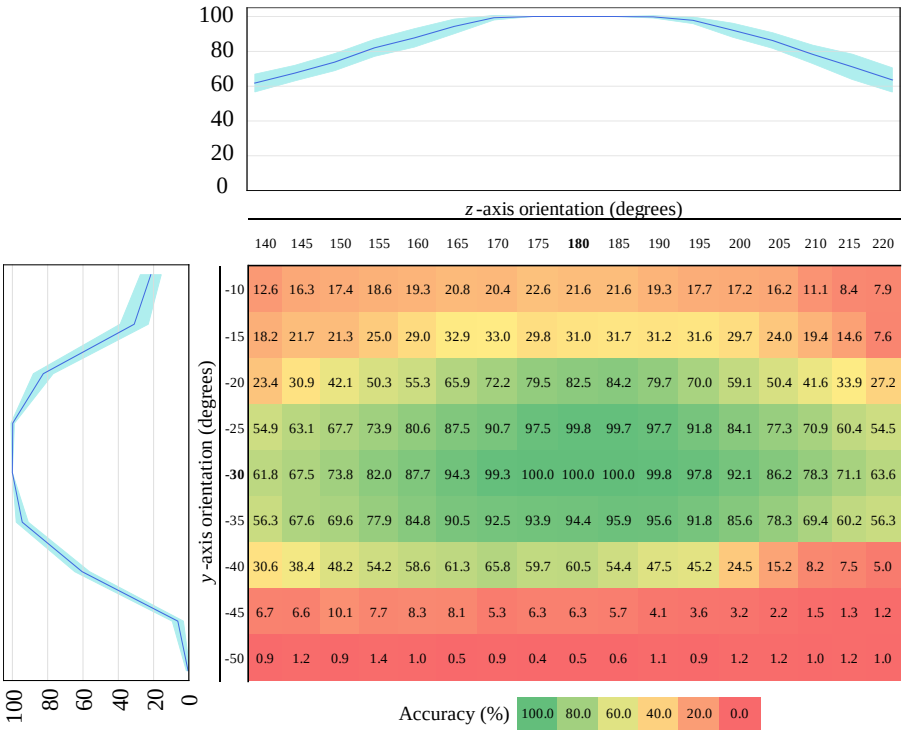

Figure 8: Heat map of the BM evaluation accuracy. The percentage shown for each camera pose is the average of 1,000 test episodes. The orientations in bold on the axes correspond to the pose of the camera during training. The top graph depicts the average accuracy across the z-axis. It represents the projection of the heat map when it intersects a plane perpendicular to the y-axis at $-30°$. The left plot shows the accuracy across the y-axis. It is the projection of the heat map's intersection with a plane perpendicular to the z-axis at $180°$. The shade to the sides of the lines corresponds to ± 1 standard deviation.

The accuracy around the z-axis is almost symmetrical for all values of y except at $y = 40°$, where the accuracy at $z \in [140°, 170°]$ and $z \in [190°, 220°]$ is clearly different. We believe that this behavior is due to the effect of the robot's shadow, which may reduce the contrast between the target and the floor. The same effect can be observed, albeit with less intensity, on the top and bottom rows of Figure 8. Apparently, the agent responds better to pictures taken from the robot's left side, in which the shadow is behind the relevant elements in the scene.

The agent's behavior is not symmetrical along the y-axis either. The performance for the worst-case scenarios is considerably better when the camera is positioned below the robot's middle plane at $-10°$ than when it is above it at $-50°$. Figure 6 suggests that the agent may have struggled to properly infer the joint positions at $-50°$, since some links may have been hidden. However, at $-10°$, the challenge is to estimate depth because the camera orientation is nearly parallel to the floor plane. In light of the results, it seems easier to handle the lack of depth data than with the loss of information about the joint positions.

5.2 Domain Randomization Model (DRM)

5.2.1 Training

Figure 9 displays the training curve for the DRM. In this case, the steady-state regime is reached at around 40 million steps, which is consistent with the non-negligible increment in the problem complexity. The agent exploited more sub-optimal policies than in the BM, the most outstanding at approximately 50 and 65 million steps. This could be noise introduced by the randomized camera location. The training time was 15 hours.

5.2.2 Robustness

Owing to the spread experience, the agent maintains accuracy above 90% for almost all the orientations between $z \in [155°, 210°]$ and $y \in [-40°, -25°]$, which is a clearly larger range than that obtained in the BM. 100% accuracy is achieved in 11 poses, most of them at $y = -30°$ and within the interval $z \in [165°, 200°]$. The maximum failure distance is 81 cm at $(140°, -10°)$, one of the farthest locations from those considered in the training phase.

The average accuracy around the z-axis is more symmetrical than in the BM, especially in the intermediate y angles. Despite the 20% and 15% accuracy drop at $-45°$ and $-15°$, respectively, the agent learned to manage the effect of

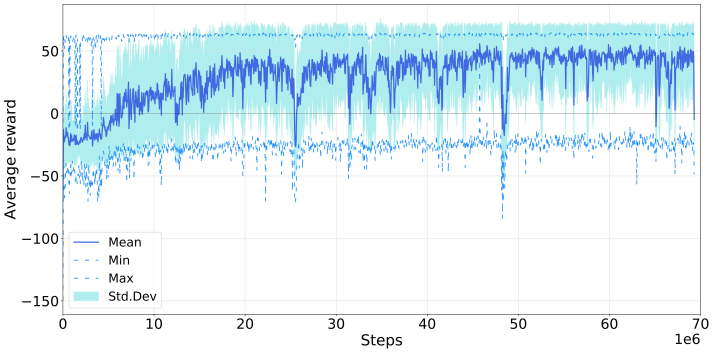

Figure 9: DRM training curve across 70 million steps. The x-axis is the number of steps, whereas the y-axis is the average value obtained during the training evaluation interval. Each of the points on the plot is the average performance achieved. The steady-state regime, with an average reward of 40, begins around 40 million steps. Note that the DR learning curve is more irregular than the one obtained for the BM, likely due to noise and the problem's complexity. In this case, the valleys caused by the sub-optimal policies are registered at 50 million and 65 million steps.

the shadows quite well. Overall, the results are definitely better. Nonetheless, it is interesting to point out that the slopes of the projections are basically parallel.

When the camera pose changes around the y-axis, DRM performance at $-10°$ decreases compared to the BM, whereas it is higher at $-50°$. Since the number of training steps is kept constant, in the DRM the experience samples are distributed over a wider range. The DR agent sees more different situations, but each of them less frequently. This could explain why the Baseline agent, which is exposed to a single camera configuration for the 70 million steps, manages to cope slightly better with limited depth information. Unlike in the BM, the lack of symmetry at the extremes of y suggests that, with diverse enough samples, learning how to deal with hidden joint orientations is simpler than understanding depth.

To conclude, Figure 11 depicts the difference in DRM percentage points with respect to the BM in the heat map format. For orientations near

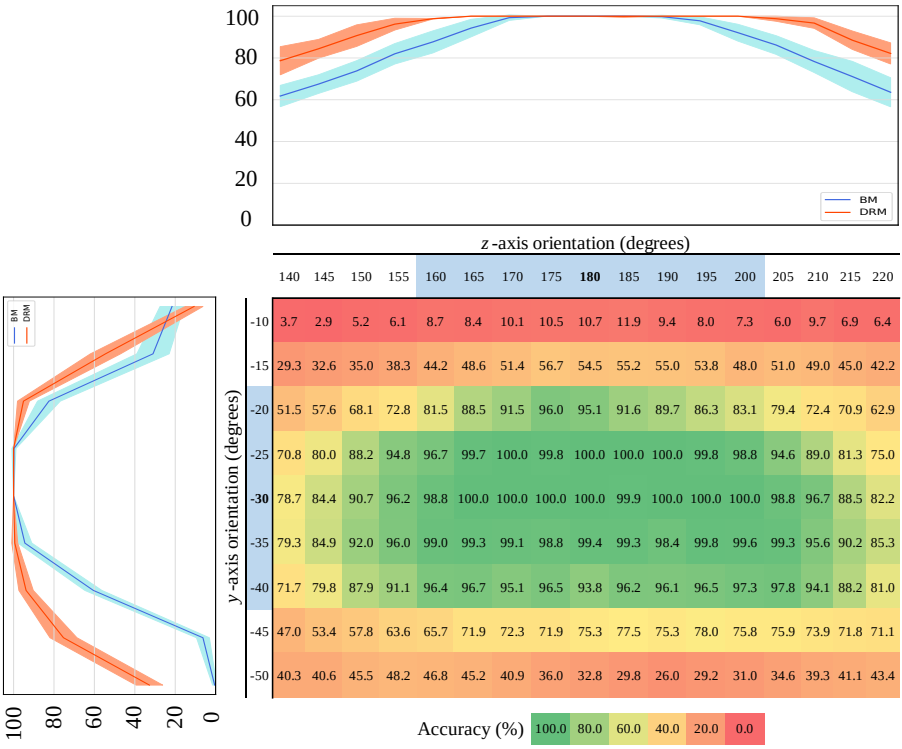

Figure 10: Heat map of the DRM evaluation accuracy. The percentage shown for each camera pose is the average of 1,000 test episodes. The orientations shown to the DR agent in the training phase are highlighted in blue. The top graph depicts the average accuracy across the z-axis. It represents the projection of the heat map when it intersects a plane perpendicular to the y-axis at $-30°$. The left plot shows accuracy across the y-axis. It is the projection of the heat map's intersection with a plane perpendicular to the z-axis at $180°$. The shading on the sides of the lines corresponds to ± 1 standard deviation.

$(180°, -30°)$ there is a tie with close to perfect accuracy. Everywhere else, the DRM enhances the accuracy outcomes significantly, exceeding 80% in some scenarios and improving accuracy around 25% on average. The y-axis projection has an asymmetrical shape due to the decrease in the DRM's accuracy at $-10°$ and to the outstanding results achieved at $-40°$, where the BM was apparently affected by the robot's shadow. Hence, it is safe to state that the DRM outperforms the BM with the same training steps (i.e., with

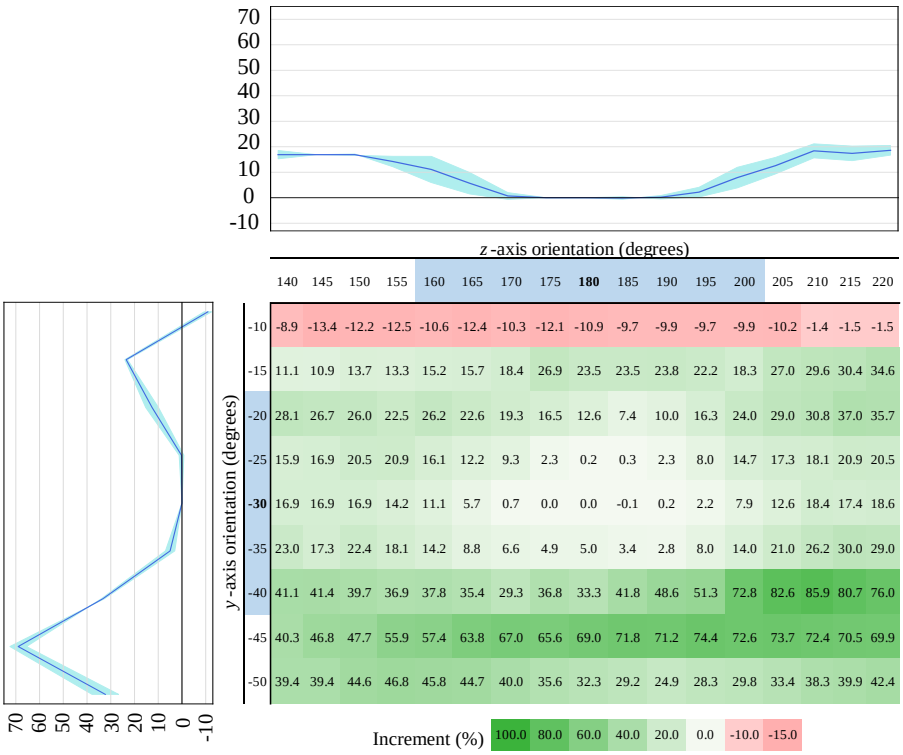

Figure 11: Difference between the average accuracy of the DRM with respect to the BM. The values shown are the average of 1,000 test episodes. On its axes, the orientations shown to the DR agent are marked in blue and in bold the single orientation used in the BM. On the top, the increment of the average accuracy achieved in evaluation across the z-axis is displayed. It represents the projection of the heat map values when it is intersected by a plane perpendicular to the y-axis at $-30°$. On the left, the increment of the average accuracy achieved in evaluation across the y-axis is depicted. It represents the projection of the heat map values when this one is intersected by a plane perpendicular to the z-axis at $180°$. The line's shading corresponds to ± 1 standard deviation of the accuracy distribution per orientation pair.

the same effort), despite being exposed to a more variable and, therefore, more complex problem.

6 Conclusion and future work

This paper compares the performance of an artificial agent trained with domain randomization (DR) with respect to a baseline designed following a mainstream PNN approach to teach a virtual robotic arm how to reach a picking area. The inputs are exclusively the images provided by an externally-mounted camera. The robustness of both models is assessed by changing the camera orientation along two axes.

The results show that right before being transferred to reality in the sim-to-real pipeline (i.e., after the virtual training phase), the DR agent is more robust to moderate changes in the setup than the baseline PNN, at the expense of a slightly higher computational effort, if any. The conclusion holds even if two camera angles are changed simultaneously. This finding suggests that applying DR in the virtual environment should boost the vanilla PNN performance and reduce the amount of experience required when transferring this knowledge to the real world, which will never exactly replicate the virtual setup.

Shifting to the physical world leads to a loss of control over the influence of the environment due to the considerable amount of new visual features that need to be monitored. Our work's core analysis and contributions focus on the part of the model that processes synthetic experience. We believe that the results are more clearly conveyed when they are not mixed with the complexities derived from adaptation to the physical setup, such as the different alternatives to implement the lateral connections of the PNN framework. Therefore, we leave for future research the quantification of the reduction of the real experience required to make agents fully operative, and the point where adding more randomization starts yielding only marginal benefits.

Author contributions

Lucía Güitta-López: Conceptualization and design of this study, Methodology, Formal analysis and investigation, Software, Data curation, Writing - original draft preparation, Writing - review and editing. **Jaime Boal**: Conceptualization and design of this study, Methodology, Formal analysis and investigation, Supervision, Writing - review and editing. **Álvaro J. López-López**: Conceptualization and design of this study, Methodology, Formal analysis and investigation, Supervision, Writing - review and editing.

Declarations

- Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.
- Conflict of interest/Competing interests: All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.
- Ethics approval: Not applicable
- Consent to participate: Not applicable
- Consent for publication: Not applicable
- Availability of data and materials: Researchers or interested parties are welcome to contact the corresponding author L.G-L. for further explanation. Data is available at: https://gitlab.com/comillas-cic/sim-to-real/pnn-dr.git
- Code availability: Researchers or interested parties are welcome to contact the corresponding author L.G-L. for further explanation. Code is available at: https://gitlab.com/comillas-cic/sim-to-real/pnn-dr.git

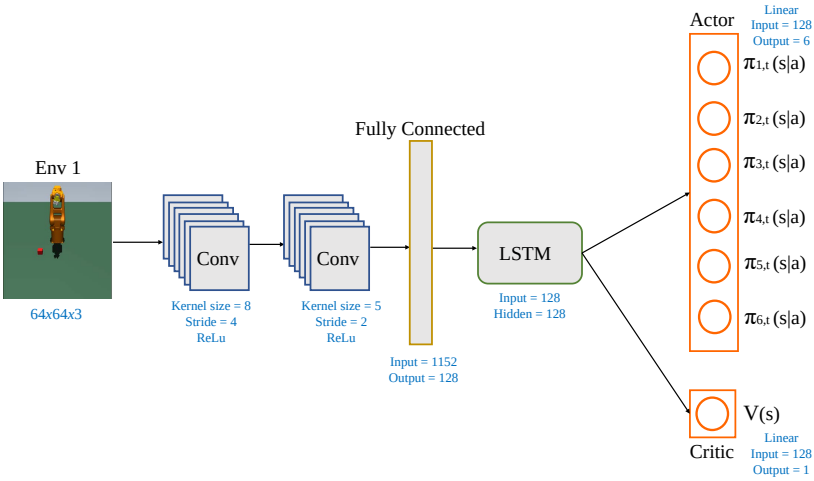

Figure A1: A3C architecture implemented. The input is a 64 x 64 RGB environment image that goes through two convolutional layers, the first defined by a 3 x 3 kernel and `stride` = 4, and the second with a 5 x 5 kernel and `stride` = 2. The model block is a fully connected layer with 1152 inputs and 128 outputs. Finally, a Long-Short Term Memory (LSTM) network with 128 hidden states is applied to better capture the sequence of movements.

Appendix A Design of the Baseline Model (BM)

The problem tackled in this paper is formulated as a fully observable MDP with an A3C agent, whose architecture replicates the proposal by Rusu et al. [4] (Figure A1). The agent's observation (i.e., the model input) is a 64 x 64 RGB rendered image of the virtual environment. There are seven outputs, six from the Actor, the policies applied to each joint (i.e., the discrete probability functions that assign the probability of applying a given option from the action set to change the joints' positions) and one for the Critic, which determines the value of the value-state function. Conversely to [4], the agent commands the position of the actuators rather than their speed, because this variable can be more easily controlled in commercial industrial robots. A softmax function is used to compute the actions' likelihood. The optimizer employed is RMSProp with a learning rate of 1×10^{-4} and a decay of 0.99.

To analyze the impact of the action set and the reward on the agent's behavior, several combinations of these hyperparameters were designed and trained (Table A1). The evaluation of each model was carried out running 1000 episodes under ten different seeds using initially 5 cm as reward distance, and then 10 cm.

The Baseline Model selected for the experiments conducted in this paper is `M1` because the results obtained are on a par with those presented in [4] in terms of the average accuracy, maximum failure distance, and learning time. It is characterized by a logarithmic action set that enables the agent to approach the target faster when it is distant and operate precisely in the area surrounding the goal. A discontinuous function establishes a negative reinforcement if the goal is not reached and a positive reinforcement otherwise.

The conclusion from these experiments is that even though action space discretization can be as granular as desired, simpler spaces lead to better results. In addition, a logarithmic action space seems to behave better than a linear approach, which could be explained by the fact that with the same number of choices, the logarithmic approach provides a better combination of coarse and fine movements.

Table A1: MDPs considered to select the Baseline Model. The chosen option is highlighted in gray.

Model	Action space[a]	Positive reward per step[b]	Negative reward per step[c]
`M0`	0 ±MPI ±MPI/2	70	$-(2 \cdot dist)^2$
`M1`	0 ±MPI ±MPI/10 ±MPI/100	70	$-(2 \cdot dist)^2$
`M2`	0 ±MPI ±MPI/10 ±MPI/100	90 if `el` $\in [0, 30]$ 70 if `el` $\in (30, 35]$ 50 if `el` $\in (35, 40]$ 30 if `el` $\in (40, 50]$	$-(2 \cdot dist)^2$
`M3`	0 ±MPI ±MPI/10 ±MPI/100	100 if `el` $\in [0, 30]$ 50 if `el` $\in (30, 35]$ 30 if `el` $\in (35, 40]$ 20 if `el` $\in (40, 50]$	$-(2 \cdot dist)^2$
`M4`	0 ±MPI ±MPI/2 ±MPI/4 ±MPI/16 ±MPI/64 ±MPI/128	70	$-(2 \cdot dist)^2$
`M5`	if distance > 1.5*`reward_dist` ±(0, MPI, MPI/2, MPI/4) if distance < 1.5*`reward_dist` ±(0, MPI/10, MPI/50, MPI/100)	70	$-(2 \cdot dist)^2$
`M6`	if distance > 1.3*`reward_dist` ±(0, MPI, MPI/2, MPI/4) if distance < 1.3*`reward_dist` ±(0, MPI/4, MPI/10, MPI/25)	90 if `el` $\in [0, 30]$ 70 if `el` $\in (30, 35]$ 50 if `el` $\in (35, 40]$ 30 if `el` $\in (40, 50]$	$-(2 \cdot dist)^2$

[a] MPI stands for Maximum Position Increment and `reward_dist` for rewarding distance.
[b] `el` is an abbreviation for episode length.
[c] $dist$ is the distance between the gripper and the target.

References

[1] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. The MIT Press, Cambridge (Massachusetts), London (England) (2018)

[2] Mahmood, A.R., Korenkevych, D., Vasan, G., Ma, W., Bergstra, J.: Benchmarking reinforcement learning algorithms on real-world robots. In:

Proc. 2nd Conf. Robot Learning, vol. 87, pp. 561–591 (2018)

[3] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning. Foundations and Trends in Machine Learning **11**(3–4), 219–354 (2018). https://doi.org/10.1561/2200000071

[4] Rusu, A.A., Večerík, M., Rothörl, T., Heess, N., Pascanu, R., Hadsell, R.: Sim-to-real robot learning from pixels with progressive nets. In: 1st Conf. Robot Learning (2017)

[5] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp. 23–30 (2017)

[6] Bellman, R.: A Markovian decision process. Journal of Mathematics and Mechanics, 679–684 (1957)

[7] Monahan, G.E.: Survey of partially observable Markov decision processes - Theory, models and algortihms. Management Science **28**(1), 1–16 (1982). https://doi.org/10.1287/mnsc.28.1.1

[8] Al-Masrur Khan, M.D., Khan, M.R.J., Tooshil, A., Sikder, N., Parvez Mahmud, M.A., Kouzani, A.Z., Nahid, A.A.: A systematic review on reinforcement learning-based robotics within the last decade. IEEE Access **8**, 176598–176623 (2020). https://doi.org/10.1109/ACCESS.2020.3027152

[9] Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous methods for

deep reinforcement learning. In: Proc. 33rd Int. Conf. Machine Learning, vol. 48, pp. 1928–1937 (2016)

[10] Gu, Z., Jia, Z., Choset, H.: Adversary A3C for robust reinforcement learning. In: Int. Conf. Learning Representations (2018)

[11] Grondman, I., Busoniu, L., Lopes, G.A.D., Babuška, R.: A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews **42**, 1291–1307 (2012). https://doi.org/10.1109/TSMCC.2012.2218595

[12] Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., Kautz, J.: Reinforcement learning through asynchronous advantage actor-critic on a GPU. In: Int. Conf. Learning Representations (2017)

[13] Lazaridis, A.: Deep reinforcement learning: A state-of-the-art walk-through. Journal of Artificial Intelligence Research **69**, 1421–1471 (2020)

[14] Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine **34**(6), 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240

[15] Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539

[16] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, - (2016)

[17] Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. International Journal of Robotics Research **32**(11), 1238–1274

(2013). https://doi.org/10.1177/0278364913495721

[18] Zamfirache, I.A., Precup, R.E., Roman, R.C., Petriu, E.M.: Reinforcement learning-based control using Q-learning and gravitational search algorithm with experimental validation on a nonlinear servo system. Information Sciences **583**, 99–120 (2022). https://doi.org/10.1016/j.ins.2021.10.070

[19] Singh, B., Kumar, R., Singh, V.P.: Reinforcement learning in robotic applications: a comprehensive survey. Artificial Intelligence Review **55**, 945–990 (2022). https://doi.org/10.1007/s10462-021-09997-9

[20] Zhao, W., Queralta, J.P., Westerlund, T.: Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In: IEEE Symposium Series on Computational Intelligence, pp. 737–744 (2020). https://doi.org/10.1109/SSCI47803.2020.9308468

[21] Chen, X., Hu, J., Jin, C., Li, L., Wang, L.: Understanding domain randomization for sim-to-real transfer. In: Int. Conf. Learning Representations (2022)

[22] Rusu, A.A., Colmenarejo, S.G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., Hadsell, R.: Policy distillation. In: Proc. Int. Conf. Learning Representations (2016)

[23] Wang, J.X., Kurth-Nelson, Z., Soyer, H., Leibo, J.Z., Tirumala, D., Munos, R., Blundell, C., Kumaran, D., Botvinick, M.M.: Learning to reinforcement learn. In: CogSci (2017). https://www.deepmind.com/publications/learning-to-reinforcement-learn

[24] Morimoto, J., Doya, K.: Robust reinforcement learning. Neural Computation **17**(2), 335–359 (2005). https://doi.org/10.1162/0899766053011528

[25] Hussein, A., Gaber, M.M., Elyan, E., Jayne, C.: Imitation learning: A survey of learning methods. ACM Computing Surveys **50**(2) (2017). https://doi.org/10.1145/3054912

[26] Zhu, Y., Wang, Z., Merel, J., Rusu, A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., Heess, N.: Reinforcement and imitation learning for diverse visuomotor skills. In: Proceedings of Robotics: Science and Systems (2018). https://doi.org/10.15607/RSS.2018.XIV.009

[27] Traoré, R., Caselles-Dupré, H., Lesort, T., Sun, T., Díaz-Rodríguez, N., Filliat, D.: Continual reinforcement learning deployed in real-life using policy distillation and sim2real transfer. In: Proc. Int. Conf. Machine Learning (2019)

[28] Arndt, K., Hazara, M., Ghadirzadeh, A., Kyrki, V.: Meta reinforcement learning for sim-to-real domain adaptation. In: IEEE Int. Conf. Robotics and Automation (2020)

[29] Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., Lerchner, A.: DARLA: Improving zero-shot transfer in reinforcement learning. In: Proc. 34th Int. Conf. Machine Learning, vol. 70, pp. 1480–1490 (2017)

[30] Shoeleh, F., Asadpour, M.: Skill based transfer learning with domain adaptation for continuous reinforcement learning domains. Applied Intelligence **50** (2020). https://doi.org/10.1007/s10489-019-01527-z

[31] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., Vanhoucke, V.: Using simulation and domain adaptation to improve efficiency of deep robotic grasping, pp. 4243–4250 (2018). https://doi.org/10.1109/ICRA.2018.8460875

[32] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Proc. 27th Int. Conf. Neural Information Processing Systems, vol. 27 (2014)

[33] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. In: Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 2242–2251 (2017). https://doi.org/10.1109/CVPR.2017.241

[34] Zhu, J.-Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: IEEE Int. Conf. Computer Vision, pp. 2242–2251 (2017). https://doi.org/10.1109/ICCV.2017.244

[35] Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. Computing Research Repository (CoRR) (2016)

[36] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd Int. Conf. Learning Representations (2015)

[37] James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., Bousmalis, K.: Sim-to-real via

sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In: Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, pp. 12619–12629 (2019). https://doi.org/10.1109/CVPR.2019.01291

[38] Mozifian, M., Zhang, A., Pineau, J., Meger, D.: Intervention design for effective sim2real transfer. Computing Research Repository (CoRR) (2020)

[39] Chan, S.C.Y., Fishman, S., Canny, J., Korattikara, A., Guadarrama, S.: Measuring the reliability of reinforcement learning algorithms. In: Int. Conf. Learning Representations (2020)

[40] Jordan, S.M., Chandak, Y., Cohen, D., Zhang, M., Thomas, P.S.: Evaluating the performance of reinforcement learning algorithms. In: Proc. 37th Int. Conf. Machine Learning (2020)

[41] Todorov, E., Erez, T., Tassa, Y.: MuJoCo: A physics engine for model-based control. In: IEEE Int. Conf. Intelligent Robots and Systems, pp. 5026–5033 (2012). https://doi.org/10.1109/IROS.2012.6386109

[42] Stevens, E., Antiga, L., Viehmann, T.: Deep Learning with PyTorch, (2020)